Bit Mapping for Balanced PCM Cell Programming

Yu Du, Miao Zhou, Bruce R. Childers, Daniel Mossé and Rami Melhem
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA, 15260
{fisherdu,miaozhou,childers,mosse,melhem}@cs.pitt.edu

ABSTRACT

Write bandwidth is an inherent performance bottleneck for Phase Change Memory (PCM) for two reasons. First, PCM cells have long programming time, and second, only a limited number of PCM cells can be programmed concurrently due to programming current and write circuit constraints,

For each PCM write, the data bits of the write request are typically mapped to multiple cell groups and processed in parallel. We observed that an unbalanced distribution of modified data bits among cell groups significantly increases PCM write time and hurts effective write bandwidth. To address this issue, we first uncover the cyclical and cluster patterns for modified data bits. Next, we propose *double XOR mapping (D-XOR)* to distribute modified data bits among cell groups in a balanced way. D-XOR can reduce PCM write service time by 45% on average, which increases PCM write throughput by 1.8×. As error correction (redundant bits) is critical for PCM, we also consider the impact of redundancy information in mapping data and error correction bits to cell groups. Our techniques lead to a 51% average reduction in write service time for a PCM main memory with ECC, which increases IPC by 12%.

Categories and Subject Descriptors

B.3 [**Memory Structures**]: Semiconductor Memories; C.4 [**Performance of Systems**]: Design Studies

General Terms

Design, Performance

Keywords

Phase-change memory, memory write performance

1. INTRODUCTION

Phase change memory (PCM) is a promising non-volatile memory technology that has several advantages. First, data can be stored without refresh operations, allowing PCM chips to be powered off when not in use, to save energy. Second, unlike NAND flash memory, PCM is bit-addressable and has a read access latency comparable to DRAM. Third, PCM is scalable: an 8Gb PCM chip has been demonstrated in 20nm process technology [3]. Despite promise, PCM has weaknesses, which are the subject of much research. Wear-leveling techniques have been proposed to evenly distribute writes to PCM memory lines [35, 21, 23]. Fault-tolerance techniques have been proposed to protect PCM chips from weak cell failures [22, 19, 31, 8, 9].[1] Write-pausing has been proposed to reduce the performance penalty from long PCM write service time [18].

Despite advancements, limited write bandwidth remains a major performance bottleneck for PCM. Write operations (a.k.a. *cell programming*) in PCM require complicated control circuits, long service time and high energy (to heat and cool a cell). For example, with a typical power supply, the write bandwidth of Samsung's 8Gb PCM chip [3] is only 40MB/s, while the write bandwidth of a typical DDR3 DRAM chip is 1.6GB/s. To improve PCM write throughput, circuit-level techniques typically reduce the number of cells that must be programmed (e.g., differential write [35], Flip-N-Write [2] and row buffer designs [13]). Fine-grained write power management has also been proposed to improve PCM write throughput under a fixed power budget [6, 10]. In this paper, we show that further opportunities exist to increase PCM write throughput.

The data bits of a PCM write request are mapped to multiple *cell groups*; groups are processed in parallel [26]. Due to limits on programming current, a PCM device cannot simultaneously program all data bits in a cell group [12]. Therefore, the programming time of a cell group is determined by the number of modified bits in the group. Further, the write service time is determined by the cell group that has the longest programming time. To reduce the write service time, it is important to find a proper mapping between data bits and PCM cell groups to spread the number of modified data bits among all groups.

We characterized the distribution patterns of modified data bits to understand how bits are changed by a typical program. From the set of workloads that we studied, we found two common patterns: a *cyclical pattern* in which bits at a fixed distance from one another are likely to change together and a *cluster pattern* in which bits near a modified bit are also likely to be modified.

[1]Weak cells have much lower write endurance than other PCM cells.

ISCA '13 Tel-Aviv, Israel

Based on these observations, we propose new XOR mapping functions between data bits and cell groups. These functions can be used to distribute modified data bits with both cyclical and cluster patterns evenly to cell groups. Due to better balance among writes to cell groups, the write service time is reduced, which improves PCM write throughput. Additionally, we propose *double XOR mapping (D-XOR)* that uses a single mapping function to consistently generate balanced distribution of modified bits for different cell group sizes. Our results show that D-XOR reduces PCM write service time by 45% on average, leading to a 1.8× improvement in write throughput. Lastly, because error correction is critical for PCM [26, 11], we propose a *bit swap function (BS)* to interleave data bits with ECC bits. An overall 51% reduction in write service time with D-XOR and swapping leads to a 12% average IPC improvement over Flip-N-Write for a PCM main memory with ECC.

This paper makes the following contributions:

- It shows that the mapping between data bits and PCM cell groups has a great impact on PCM write service time.
- It characterizes the distribution patterns of modified data bits inside memory write requests.
- It proposes new XOR mapping functions to evenly distribute modified data bits to PCM cell groups.
- It proposes bit swap functions to extend our techniques for PCM with error correction bits.

The rest of the paper is organized as follows. In Section 2, we describe the problem of mapping data bits to PCM cell groups and the impact on write service time. In Section 3, we characterize the distribution patterns of modified bits inside memory write requests. In Section 4, we describe XOR-based mapping and bit swap functions. In Section 5, we describe our experimental setting and workloads, and in Section 6, we present simulation results. Sections 7 and 8 summarize related work and conclusions.

2. DATA-TO-CELL MAPPING PROBLEM

2.1 Write Programming in PCM Devices

Unlike charge-based DRAM, PCM uses different resistance states of phase change material to represent data. A PCM write operation is more complex than a DRAM write operation. Specifically, for each write request, the data is statically divided into cell groups and distributed to multiple chips. Each cell group is independently processed. For each cell group, the write control logic first reads the old data of the request. It next uses a finite-state machine (FSM) to write data bits whose old and new values differ [29]. Precisely-controlled SET and RESET pulses are used to program PCM cells. For single-level cell (SLC) PCM, it takes 50-100ns for RESET programming (1→0) and 150-400ns for SET programming (0→1) [3, 14].

Programming current is the major constraint which limits the number of cells that can be concurrently programmed per chip [12]. For example, Samsung's 20nm 8Gb PCM chip supports only simultaneous programming of 128 cells with the default power supply [3]. Therefore, PCM devices have asymmetric read and write data widths. For PCM reads, all data bits are read concurrently, while for PCM writes, the

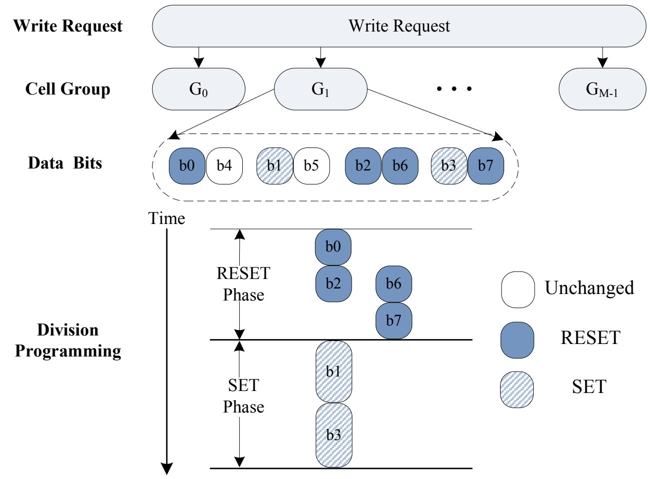

Figure 1: Division program operations.

modified bits of each cell group are written using *division program operations* [12]. For each division program operation, a subset (i.e., a *division*) of memory cells among a cell group are programmed rather than all cells in the group. Figure 1 shows an example of 2X division program operation [12]. For each cell group, the data bits are statically divided into divisions. 2X means that each division consists of two cells. In this example, a 8-bit cell group is divided into four divisions: <0, 4>, <1, 5>, <2, 6> and <3, 7>. Cells in the same division can be concurrently programmed, while cells in different divisions must be programmed in different iterations.

Divisions in the same cell group are programmed in a fixed sequential order. A division will be skipped if it has no cell to program. The programming process is divided into two phases: a RESET phase and a SET phase. In the RESET (SET) phase, only cells that need RESET (SET) programming are programmed. In Figure 1, cells 2 and 6 are programmed concurrently because they belong to the same division. Cells 3 and 7 are programmed separately because they have different programmed states. With division program operations, the programming time of each cell group is no longer a constant. On average, a cell group with more modified bits requires more programming iterations and tends to have a longer programming time. For each write request, the write service time depends on the cell group that takes the longest programming time.

2.2 Mapping Data Bits to PCM Cells

With division program operations, the mapping function (abbreviated as *mapping*) between data bits and PCM cells will impact the write service time. To illustrate the importance of the mapping, Figure 2 shows an example in which the size of the write request is sixteen bits, seven of which are modified. In this example, the sixteen data bits are mapped to four cell groups. Figure 2(a) shows a possible mapping in which adjacent data bits are directed to the same group. Notice that bits 4-7 are adjacent and modified. They are mapped to the same cell group and cause a bottleneck on write service time for the write request. Figure 2(b) shows an alternate mapping in which adjacent data bits are distributed to different cell groups. With this new mapping, the

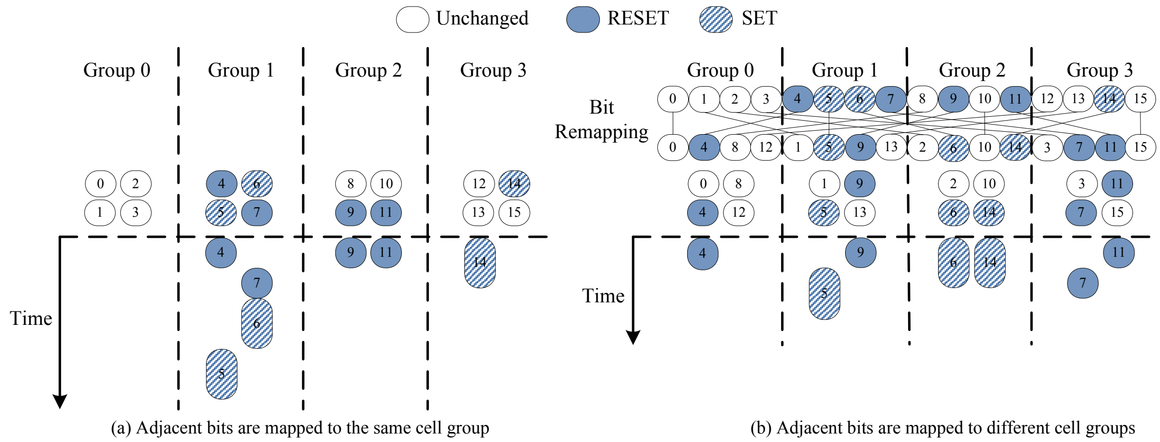

Figure 2: Example mapping between data bits and PCM cells that affects write service time.

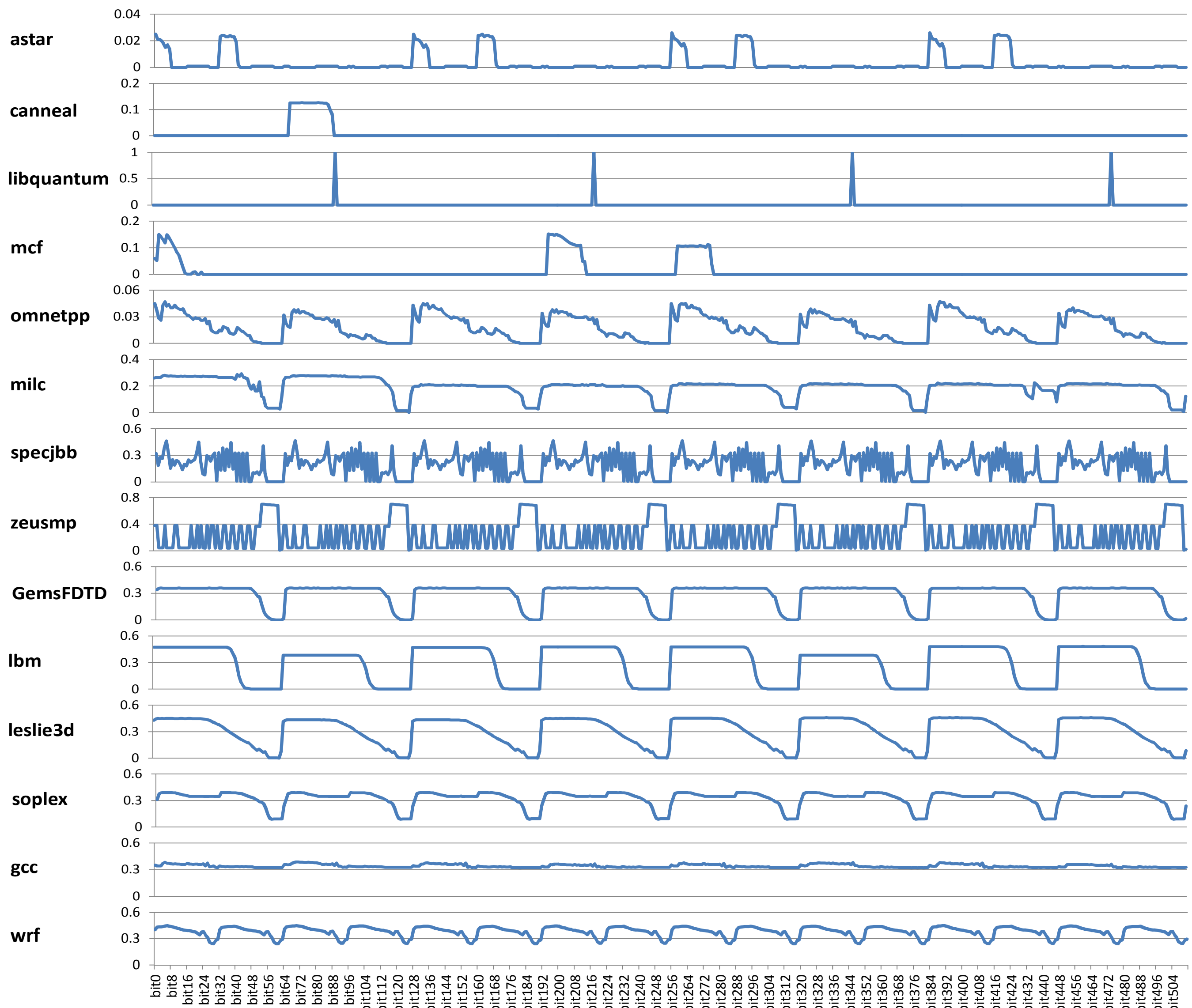

Figure 3: Average bit flip rate at different bit positions (first 512 bits of 256-byte memory requests).

four adjacent modified bits are distributed more uniformly among cell groups and the write service time is reduced. The intent of this example is not to show that the second mapping is always better than the first one, but to show that the mapping has an impact on the write service time. A good mapping should spread the modified bits among cell groups, which depends on the distribution of the modified bits in the write request.

3. DISTRIBUTION OF MODIFIED BITS

Before examining mapping functions, we consider the patterns of how modified bits are distributed in write requests to help guide the design of effective mapping functions.

We characterize *bit flip rate* to understand the distribution (patterns) of modified bits in write requests. Bit flip rate at a given bit position is defined as the probability that the bit at that position changes its state (flips) on a memory write. To calculate the bit flip rate at position K, the number of flips in the K^{th} bit of all write requests is counted and divided by the total number of write requests. We characterized all bit positions in the memory write traces (after a 32MB DRAM cache) for fourteen applications.[2] Figure 3 shows the bit flip rate histogram of the first 512 bit positions; the remaining positions are similar. We assume that all PCM writes are 256 bytes wide. Applications are ordered by the average percentage of modified data bits per write request in ascending order.

As the figure shows, the bit flip rate histograms differ for each application. We identified two general patterns in the bit flips: *cyclical* and *cluster*.

Cyclical Pattern. In this pattern, the flip rate histogram shows cyclical fluctuations. As shown in Figure 3, a typical cycle length can be 32 bits, 64 bits or 128 bits (*mcf* has a cycle length of 512 bits). Bit positions with the same low address bits but different high address bits tend to have similar bit flip rates. To distribute modified data bits with a cyclical pattern, the high address bits should be used as an index to map the bits to cell groups. For example, in application *libquantum*, data bits with the same low 7-bit address should be mapped to different cell groups.

Cluster Pattern. Modified bits may also tend to aggregate into clusters. For example, in Figure 3, *astar*, *canneal* and *mcf* have most of their modified bits clustered in only a few bytes. To understand the cluster pattern at byte granularity, we characterize the average number of bytes that cover most modified bits of each memory write (choosing those bytes that have the most number of modified bits). Figure 4 shows the average number of bytes with 90% coverage. From the figure, *astar*, *mcf*, *omnetpp* and *canneal* have strong clusters. The remaining applications show moderate clusters, with 20% to 60% of the bytes covering 90% of the modified bits. *Libquantum* is a special case. It requires a few bytes to cover most modified bits because it only has one bit modified per sixteen bytes. When clustered together, modified bits have the same high address bits but different low address bits. A good mapping function should use low address bits to distribute adjacent bits to different cell groups. For example,

[2]We use twelve write-intensive benchmarks from SPEC CPU2006, SPEC JBB2005 and the only write-intensive benchmark (*canneal*) from the PARSEC suite.

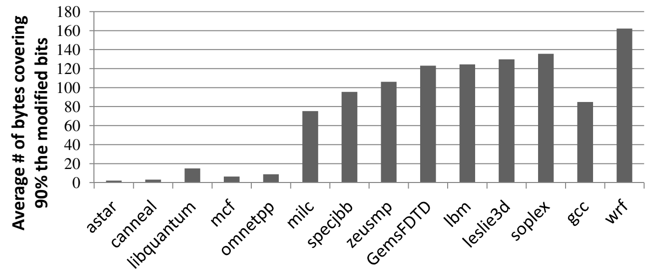

Figure 4: Average number of bytes covering 90% of the modified bits of 256-byte write requests.

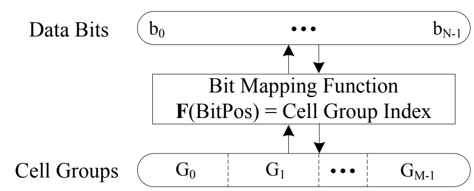

Figure 5: Bit mapping function.

in *canneal*, adjacent data bits should be mapped to different cell groups.

The distribution of modified data bits is affected by data structure and data type used by the program. Programs whose modified bits have a cluster pattern tend to update only some data object fields, possibly even just a part of a field. These programs usually have more random object access. Programs whose modified bits have a cyclical pattern often update specific fields in arrays of objects. Programs dominated by integer types tend to modify adjacent bits within an integer, while programs dominated by floating point (FP) types modify bits with a more random distribution within the float. Also, FP programs typically modify more bits per write. These behaviors are due to integer and FP encoding, and how the types are used. For example, *libquantum* has an array of objects with an integer flag. One bit in the integer is updated frequently, causing spikes as seen in Figure 3. The flip rates for other programs can be similarly explained according to data structure and type.

4. BIT MAPPING FUNCTION

4.1 Problem Definition

In this paper, we do not study the data bits to cell mapping inside each cell group: we assume that the order of mapped cells inside a cell group is the same as the order of data bits. Therefore, the problem is to find a *mapping function* F that maps N data bits from write requests to M PCM cell groups (N/M cells per cell group), while minimizing the imbalance in the number of modified data bits among all cell groups. The concept is shown in Figure 5.

The input of the mapping function F is a n-bit binary address $a_{n-1}a_{n-2}...a_0 (n = \log N)$ representing the bit position of each data bit. The output of the mapping function F is an m-bit binary number $t_{m-1}t_{m-2}...t_0 (m = \log M)$ representing the index of the cell group to which the data bit is mapped. A constraint on F is that the number of data bits mapped to each cell group should be equal. Otherwise, some cell groups will not have enough PCM cells to establish a 1:1

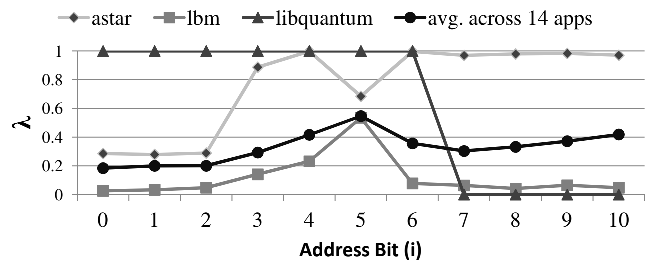

Figure 6: Distribution imbalance of modified bits at different address bits.

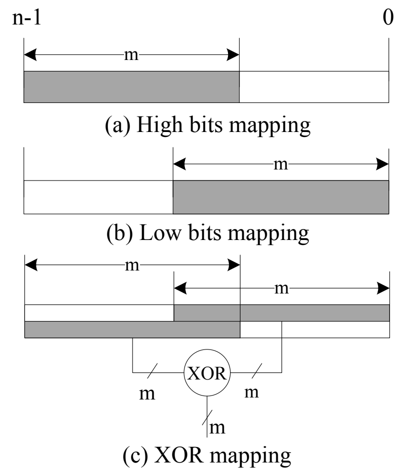

Figure 7: XOR mapping function.

mapping between data bits and cells. A mapping function defines a static partition of N data bits into M groups of equal size. We need to find a partition that can minimize the longest programming time among all cell groups. Since over the long term, a modified bit is equally probable to be SET or RESET, we approximate the problem to minimize the imbalance in the number of modified data bits among all cell groups, which is achievable with a better mapping function.

4.2 Distribution Imbalance of Modified Bits

To determine which address bits should be used in the mapping function, we characterize the distribution imbalance of modified bits at different address bits for a data bit position. For the i^{th} address bit, modified bits can be divided into two subsets:

$S_0 = \{b_k, k = a_{n-1}, \cdots, a_{i+1}, 0, a_{i-1}, \cdots, a_0\}$ and

$S_1 = \{b_k, k = a_{n-1}, \cdots, a_{i+1}, 1, a_{i-1}, \cdots, a_0\}$.

S_0 includes all modified bits with the a_i bit of the bit position equal to 0 and S_1 includes all modified bits with the a_i bit of the bit position equal to 1.

Similar to [16], we use the *percent imbalance* metric, $\lambda(i)$, to characterize the distribution imbalance of modified bits at the i^{th} address bit.

$$\lambda(i) = \left(\frac{max(|S_0|, |S_1|)}{(|S_0| + |S_1|)/2} - 1 \right) \times 100\% \quad (1)$$

When $\lambda(i) = 0$, modified bits are equally distributed between S_0 and S_1. When $\lambda(i) = 1$, there is no modified bit in S_0 or S_1. We calculate the distribution imbalance of an address bit by averaging $\lambda(i)$ over all write requests of each application.

An address bit with low λ implies that the address bit is a good candidate for spreading modified bits into two balanced subsets. In general, a good mapping function should utilize address bits with low λ.

In Figure 6, we show λ for each of the eleven address bits (256-byte write request) for three representative applications: *astar*, *lbm* and *libquantum*. *Astar* has a strong cluster pattern, *libquantum* has a cyclical pattern, and *lbm* has both patterns. In the figure, it is apparent that each application has a unique λ profile. To get a balanced distribution of modified bits, different applications should ideally use different address bits in the mapping of bits to cell groups.

The figure also shows the average λ over fourteen applications. From the figure, two address bit regions, on average, have low λ. One region is $a_6 ... a_{10}$, which reflects cyclical patterns and the other region is $a_0 ... a_2$, which reflects clusters at byte granularity. This observation is consistent with the two patterns described in the previous section. To design an effective mapping function, address bits from both regions should be used.

4.3 XOR Mapping Function

From the distribution imbalance characterization, two mapping functions can be directly derived. As shown in Figure 7(a), to get an m-bit cell group index, high address bits can be used for the mapping function such that a data bit $b_i, i = a_{n-1}a_{n-2}...a_0$, is mapped to $Group_j$ where $j = a_{n-1}a_{n-2}...a_{n-m}$. When high address bits are used, adjacent data bits are mapped to the same cell group and the mapping function exploits the cyclical pattern. Alternatively, as shown in Figure 7(b), low address bits can be used, that is, a data bit $b_i, i = a_{n-1}a_{n-2}...a_0$, is mapped to $Group_j$ where $j = a_{m-1}a_{m-2}...a_0$. When low address bits are used, adjacent data bits are mapped to different cell groups, and the mapping function exploits the cluster pattern. However, each of these mapping functions has a limitation that only m bits out of n address bits are utilized.

As shown in Figure 7(c), we devised an *XOR mapping function* that combines low address bits with high address bits. This way, either pattern is incorporated seamlessly into the mapping. Similar to the high address and low address mappings, this new mapping distributes an equal number of data bits to each cell group. This is a necessary feature of any valid mapping function, which can be verified by calculating the cell group index of each bit position.

4.4 Support Multiple Cell Group Sizes

As shown in Section 6, an XOR mapping function is optimized for a specific cell group size. There are scenarios where a single mapping function is needed for multiple cell group sizes. For example, server memory configurations typically map each request to more memory chips (more cell groups) to improve reliability [4]. Also, a PCM chip can have a wider programming width (smaller cell group size) in the SET phase than in the RESET phase [12, 32] because the programming current of a SET pulse is much smaller than the programming current of a RESET pulse.

We use an example to illustrate why a single XOR mapping function is not the optimized solution in these scenarios. Figure 8(a) shows XOR mapping functions that are size

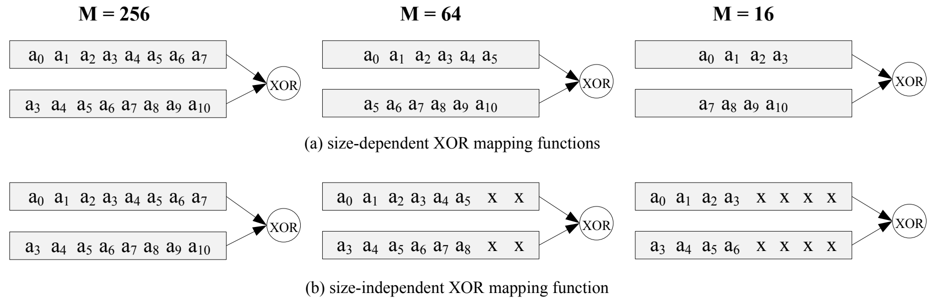

Figure 8: XOR mapping functions for different number of cell groups, M; example with eleven address bits and $M = 16, 64, 256$.

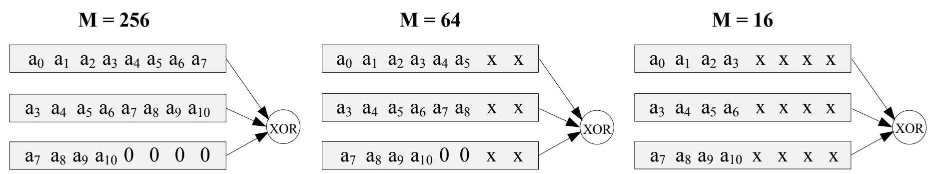

Figure 9: D-XOR mapping function for variable , M; example with eleven address bits and $M = 16, 64, 256$.

dependent. For each unique number of cell groups, M, a specific implementation of XOR mapping is used. Figure 8(b) shows how a single function can be used to support mapping different number of cell groups. The XOR mapping function is optimized for 256 cell groups. To map to fewer number of cell groups, the low bits of the mapping function are masked. The masking process is achieved by treating data bits in adjacent cell groups as a larger single cell group. Masking k bits of the mapping function means that every 2^k adjacent cell groups are treated as one group. Given that the order of data bits is unchanged, the masking process does not require any additional hardware support. The problem for masking is that fewer address bits are used in the mapping when M becomes smaller, which will degrade effectiveness.

To address this problem, we propose *double XOR mapping (D-XOR)*. In D-XOR, in addition to the XOR between low and high address bits, the high half of the high address bits is XORed with the low half of the high address bits. Figure 9 shows an example with eleven address bits. The mapping function does one XOR between $a_0...a_7$ and $a_3...a_{10}$ and a second XOR with $a_7...a_{10}$ padded with zeros. For M=16, all address bits still participate in the mapping function, although the low four bits of the function are masked.

4.5 Support PCM with Redundant Bits

To protect memory from data corruption due to transient errors, error correcting codes (ECC) are typically used in DRAM and PCM memory subsystems. Furthermore, PCM main memory typically uses a scheme to tolerate not only transient errors but also permanent errors in weak cells. To correct these errors, redundant bits are used. For typical DRAM solutions, extra DRAM chips store ECC bits [7], but for PCM, dedicated ECC chips can degrade write performance if ECC chips have more modified bits to write.

Figure 10 shows the flip rate of ECC bits normalized to the flip rate of regular data bits. The ECC code in this example is a (72, 64) SEC-DED ECC code with 8 ECC bits per 64-bit data block [7]. Since ECC is a checksum to data bits, ECC bits tend to have a higher flip rate than normal data bits.

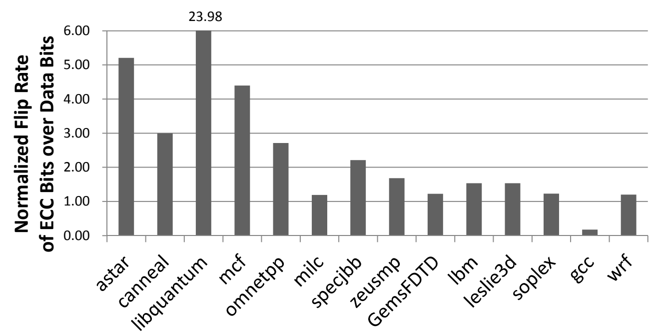

Figure 10: Flip rate of ECC bits normalized to data bits.

From the figure, in all applications, except *gcc*,[3] the ECC bits have high flip rates. For *astar*, *canneal*, *libquantum*, *mcf* and *omnetpp*, the ECC bit flip rates are much higher because most data bits are rarely modified. With a high flip rate, the PCM chips that store ECC bits have many more modified bits, which causes these redundant chips to become a write bottleneck.

Figure 11 shows our mapping function for redundant PCM storage. We use either XOR or D-XOR mapping for data bits and redundant bits. We add an extra bit swap function (BS) between regular data bits and redundant bits (e.g., ECC) to disperse the redundant bits (that have a high flip rate) among the data bits. By swapping some regular data bits with redundant bits, we avoid the situation where the cell groups for redundant storage have many more modified bits than the cell groups for data storage, which would increase imbalance.

For example, if ECC is used with a capacity ratio of 8:1 between data bits and ECC bits, then for PCM with 256-byte line size and 256-bit ECC, we pick one bit from ev-

[3] *Gcc* has many program data updates, which flips many data bits but few ECC bits.

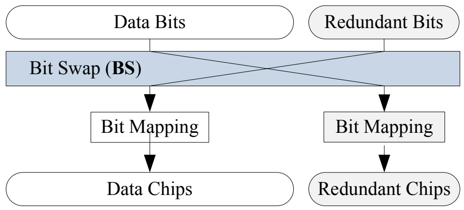

Figure 11: Mapping function for ECC memory swaps between selected data and ECC bits.

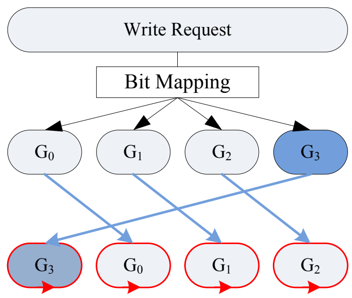

Figure 12: Revised intra-line wear leveling.

ery data byte to swap with ECC bits. To avoid selecting frequently-modified bits with a cyclical pattern, an XOR between $a_5 ... a_7$ and $a_8 ... a_{10}$ is used to determine the bit to be swapped in each data byte.

4.6 Support Intra-line Wear Leveling

There are two types of wear leveling to evenly distribute memory writes over the entire memory storage: inter-line and intra-line. Inter-line wear leveling balances memory writes among memory lines. Intra-line wear leveling balances memory writes within a memory line.

Our bit mapping functions change the bit ordering only within a memory request. Therefore, the functions are not impacted by inter-line wear leveling. For intra-line wear leveling, the conventional row shifting scheme [35] is not compatible with XOR or D-XOR mapping. Figure 12 shows two-level row shifting which is compatible with our mapping functions. First, a global row shift moves data bits at cell group granularity. Second, a local row shift moves data bits inside each cell group. With two-level row shifting, the data bits mapped to a single cell group are still mapped to a single cell group after the row shift. Also, the overhead of two-level row shifting is the same as conventional row shifting.

Intra-line wear leveling only allows for the even distribution of modified bits over the lifetime. For each write request, distribution of modified bits can still be imbalanced with intra-line wear leveling. By contrast, our mapping functions are effective in balancing the distribution of modified bits and reducing write service time for each write request.

4.7 Hardware Implementation

As shown in Section 6, a single mapping function can provide good performance for different number of cell groups. Since a mapping function is a static rearrangement of the data bits in a write request, it has negligible implementation overhead. For PCM writes, a fixed redistribution network is needed to map the data bits to the desired positions before they are sent to PCM. For PCM reads, a corresponding reverse redistribution network can restore data bits to their original positions. Changing the representation of memory data is common for PCM. For example, Zhou et al. [35] proposed a dynamic row shift mechanism to even out the writes to the cells inside a memory line. The delay and power of their 1KB row shifter are 400ps and 795μW. Our mapping functions have much lower hardware cost because they statically change only the order of data bits.

Our mapping functions do not require changes to the existing memory interface. When data bits are reordered, the memory controller divides the bits into chunks of adjacent bits and delivers each chunk to a PCM chip in the DIMM. Each chip divides the bits into cell groups of adjacent bits.

The mapping function can be implemented in the memory controller, which exposes all bits, including redundancy bits (e.g., ECC). The mapping function can also be implemented inside the PCM chips themselves, especially for redundant bits that are not visible at the memory controller (i.e., when the chip itself has a remapping or redundant bits).

CPU	8-core, 4GHz, single-issue in-order, 32KB L1 I/D
L2 Cache	private 2MB, 8-way 64-byte line size, write-back
DRAM Cache	256MB (32MB per core), 32-way 256-byte line size, write-back 30ns (120 cycles) read latency next line sequential prefetcher
Memory Controller	2 PCM channels 2 DIMMs per channel 16 banks per DIMM 32-entry write-queue per bank write pausing scheduling for PCM
Main Memory	32GB SLC PCM, 120ns read latency 100ns RESET pulse 150ns SET pulse 100ns pulse interval 32 bits per cell group, 2X division program operation

Table 1: System settings

5. EXPERIMENTAL METHODOLOGY

5.1 Configuration

We use Virtutech Simics [15] to collect main memory traces, which contain a command identifier (read or write) and address of each memory request. To accurately evaluate PCM write service time, the memory trace file also includes the value of the data before and after each memory write. The trace files are input to our own trace-driven simulator. The simulation parameters are detailed in Table 1.

We assume an 8-core 4GHz *chip multiprocessor* (CMP) with in-order cores. Each core has a 2MB private L2 cache. The shared DRAM cache is 256MB with 32MB quota per core [20]. To alleviate the cache miss penalty, we incorporated a next line sequential prefetcher for the DRAM cache.

We model a 32GB PCM main memory with two channels; each channel has two DIMMs and each DIMM has 8 chips and 16 banks. We assume 32 bits per cell group (64 cell groups per 256-byte request) and 2X division program op-

eration (up to two cells are concurrently programmed per 32-bit cell group). We use a memory controller configuration similar to Qureshi et al. [18], where each bank has a 32-entry write queue. Read requests are given highest priority, as long as the write queue is not full. For PCM memory scheduling, we use write pausing [18], whereby the memory controller suspends an active PCM write at the beginning of programming the next modified bit to schedule a higher priority read request to the memory bank. We use single-level cell (SLC) PCM to conservatively evaluate the mappings because multi-level cell PCM has much lower write bandwidth than SLC PCM. To model SLC PCM write service time, we use 100ns for cell RESET programming pulse (1→0), 150ns for cell SET programming (0→1) pulse and 100ns interval between two programming pulses [3].

Name	Reads PKI	Writes PKI	Description
Gems_r	5.35	2.14	8 copies of GemsFDTD
lbm_r	3.14	1.52	8 copies of lbm
leslie_r	3.48	0.79	8 copies of leslie3d
libq_r	8.45	1.50	8 copies of libquantum
mcf_r	11.27	6.42	8 copies of mcf
milc_r	13.42	2.58	8 copies of milc
wrf_r	0.62	0.21	8 copies of wrf
mix_1	1.34	0.37	gcc-mcf-zeusmp-canneal
mix_2	1.13	0.41	astar-gcc-gems-wrf
mix_3	4.70	0.74	libq-mcf-milc-zeusmp
mix_4	2.94	1.02	gems-leslie-mcf-zeusmp
mix_5	6.50	1.45	lbm-libq-mcf-canneal
mix_6	4.25	1.50	lbm-leslie-mcf-milc
mix_7	7.02	1.85	gems-milc-omnetpp-soplex
mix_8	8.93	1.98	libq-mcf-omnetpp-canneal
specjbb	4.03	1.05	specjbb2005, 8 warehouses

Table 2: Simulated workloads and PKIs.

5.2 Workloads

Since our work addresses the write performance bottleneck for PCM, we consider write-intensive benchmarks. We use twelve write-intensive benchmarks from SPEC CPU2006 and *canneal* from PARSEC. We use only *canneal* from PARSEC because most PARSEC benchmarks are computation-intensive or have a very small memory footprint. *Canneal* is executed in single-threaded mode and uses the native input with 940MB memory footprint. For SPEC CPU2006, we use the reference inputs. To evaluate server workloads, we also choose SPEC JBB2005. SPEC JBB2005 is a server-side Java benchmark. Our experiments use eight threads and eight warehouses.

To evaluate system performance, we choose eight representative multiprogrammed workloads, each containing two copies of four unique benchmarks; these are the mix_i workloads at the bottom of Table 2. We also ran experiments with seven applications in *rate mode*, where eight instances of the same benchmark are concurrently executed; see the top of Table 2. The other applications are not evaluated in rate mode because the 256MB DRAM cache effectively filters most main memory write requests. All benchmarks are 64-bit binaries, compiled with gcc 4.1.2. Table 2 shows the number of memory read and write requests per 1000 instructions (PKI) after the DRAM cache.

6. RESULTS

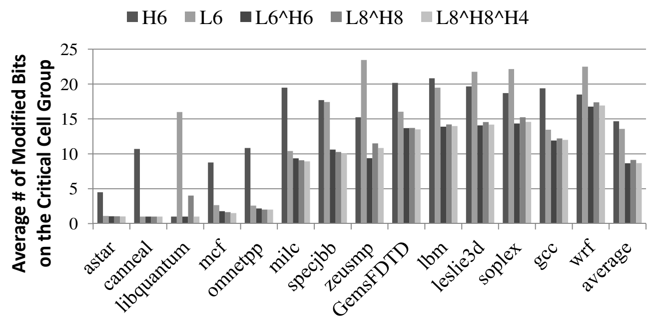

Figure 13: Average number of modified bits on the critical cell group (lower is better).

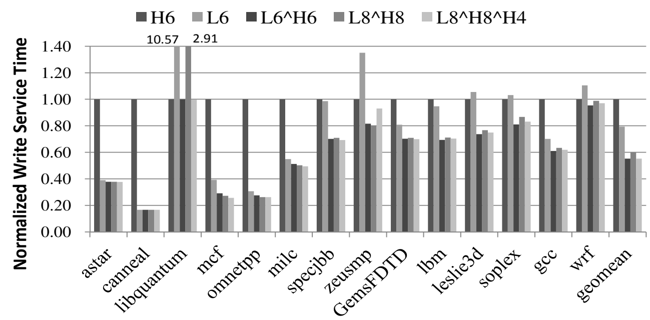

Figure 14: Average write service time normalized to H6.

This section presents simulation results for our mapping functions, with and without redundant bits. We show how performance is improved in comparison to the conventional mapping, both in terms of write service time and IPC, for each benchmark and the average over all benchmarks. Some graphs are normalized, and in the graphs, we calculate the geometric mean instead of the average to avoid the case in which one benchmark dominates the results.

We evaluated several mapping functions. Hx (Lx) uses the high (low) order x address bits for bit mapping, where x is the number of bits for the cell group index. LxˆHx is an XOR mapping function between the high x address bits and the low x address bits. LxˆHxˆHy is a corresponding D-XOR function. Unless otherwise specified, each write request is mapped to 64 cell groups (M=64, x=6). When M=64, H6 is the baseline, which maps logically adjacent 32 bits to the same cell group.

6.1 Mapping for Data Bits

Modified Bits Distribution. Figure 13 shows the average number of modified bits in the cell group that has the longest programming time for a PCM write; we call this group the *critical cell group.*[4] When modified bits are more evenly distributed among cell groups, the average number of modified bits in the critical cell group decreases, which reduces the write service time. From the figure, most applications do better with low address bits as the mapping function than

[4]If there are multiple groups having the longest programming time, a group having the maximum number of modified bits is chosen as the critical cell group.

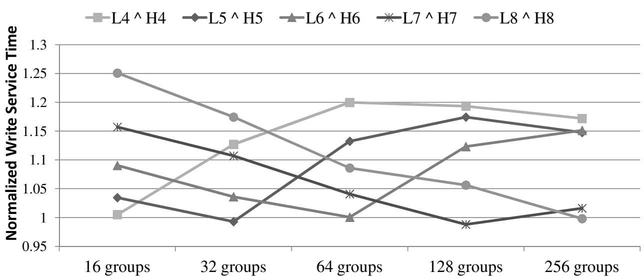

Figure 15: Average write service time of different numbers of cell groups normalized to L8ˆH8ˆH4 (averaged over all 14 benchmarks).

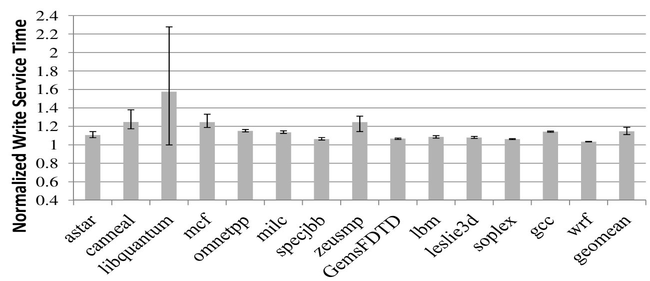

Figure 16: Average write service time of 20 random bit mapping functions normalized to L8ˆH8ˆH4.

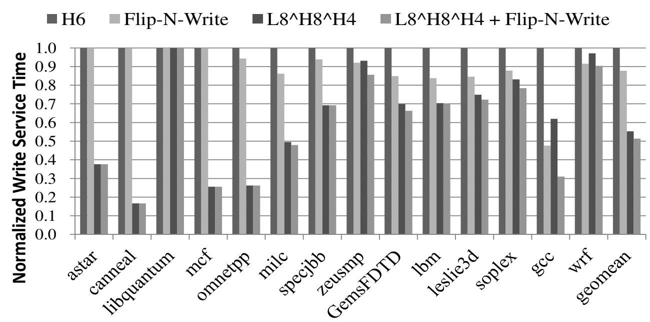

Figure 17: Comparison to Flip-N-Write.

with high address bits. This indicates that the distribution of modified bits has a strong cluster pattern. In *libquantum*, *zeusmp*, *leslie3d*, *soplex* and *wrf*, the distribution of modified bits is dominated by a cyclical pattern. For these applications, it is more effective to use high address bits as the mapping function. Overall, L6 has 7.5% fewer modified bits in the critical cell group than the H6 baseline. Because XOR functions combine both low and high address bits, they are much better than H6. L6ˆH6 has 41% fewer modified bits in the critical cell group than the H6 baseline. L6ˆH6 is slightly better than L8ˆH8 because of its effectiveness for *libquantum*. For *canneal*, address bits a_9 and a_{10} are good candidates to use in the mapping function, but these address bits are masked by L8ˆH8 when M=64. The D-XOR mapping function, L8ˆH8ˆH4, has similar results as L6ˆH6. In a later section, the advantage of D-XOR mapping for a variable number of cell groups is shown.

Write Service Time. Because divisions are sequentially programmed for each cell group, the number of modified bits in the critical cell group has a direct consequence on write service time. Figure 14 shows the average write service time of each application normalized to H6. L6, L6ˆH6 and L8ˆH8 are much better (79%, 55% and 60%, respectively) than H6. The D-XOR (L8ˆH8ˆH4) service time is similar to L6ˆH6. These results are consistent with Figure 13. The reduction in write service time also indicates an improvement in PCM write throughput. Based on these results, we conclude that, on average, L6ˆH6, L8ˆH8 and L8ˆH8ˆH4 can achieve 1.8×, 1.7× and 1.8× more write throughput than H6.

Variable Number of Cell Groups. Figure 15 shows the average write service time with different XOR mapping functions (averaged over all 14 benchmarks). All results are normalized to L8ˆH8ˆH4.

As shown in the figure, each XOR mapping function is optimized for a specific number of cell groups. If the number of cell groups is small, some high address bits are masked in the mapping function. For example, with 16 cell groups, the average write service time of L8ˆH8 is 25% worse (higher) than L8ˆH8ˆH4. If the number of cell groups is large, there is unnecessary overlap between high address bits and low address bits, which will also degrade the effectiveness of an XOR mapping function.

Overall, L8ˆH8ˆH4 has similar write service time as an optimized XOR mapping function. However, L8ˆH8ˆH4 is a *single* mapping function that adapts to variable number of cell groups, and thus, it is simpler to implement.

Comparison to Random Bit Mapping. We also compared D-XOR to the most natural shuffling, namely random bit mapping. A random mapping function is a *fixed random permutation* of data bits that eliminates any inherent distribution pattern to avoid clustering the modified bits in one cell group. Figure 16 shows the average write service time for 20 randomly-generated mapping functions. The results are normalized to L8ˆH8ˆH4 D-XOR for comparison. Because random mapping does not utilize cyclical and cluster patterns, the average write service time is 14.7% worse (higher) than L8ˆH8ˆH4. The error bars show the maximum and minimum write service time for different random instances. For most applications, the difference is small. *Libquantum* has a very large variation because the cyclical pattern allows a perfect distribution of modified bits, which cannot be achieved by most random mapping functions.

Adding Flip-N-Write. Flip-N-Write [2] is an effective technique to reduce PCM write service time. With an extra bit of storage, Flip-N-Write counts the number of bits to be written and changes the encoding of data bits to reduce the number of cells that must be programmed. Figure 17 shows the average write service time for Flip-N-Write and our L8ˆH8ˆH4 D-XOR mapping function. Note that Flip-N-Write has an extra bit flip for every 32 data bits in each cell group. All results are normalized to H6. On average, Flip-N-Write and L8ˆH8ˆH4 reduce the average write service time by 12% and 45% respectively. Since both techniques are orthogonal, applying Flip-N-Write to L8ˆH8ˆH4 further reduces the average write service time by 7% over L8ˆH8ˆH4 alone.

6.2 Mapping with Redundant Bits

Figure 18 shows the average write service time for PCM

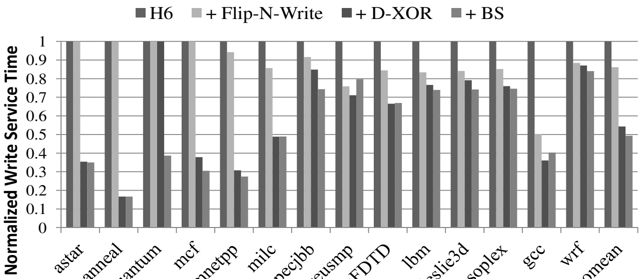

Figure 18: Average write service time for ECC memory normalized to H6.

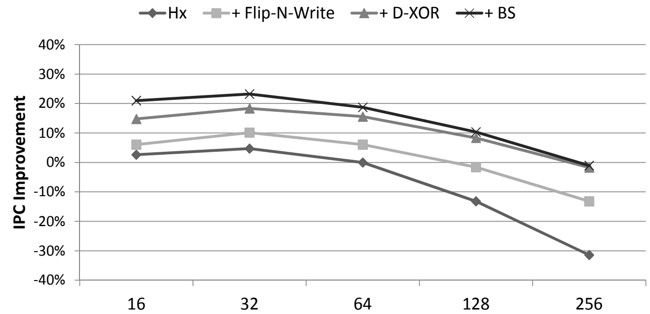

Figure 20: IPC improvement as the number of cell groups is varied (baseline is H6 with 64 cell groups).

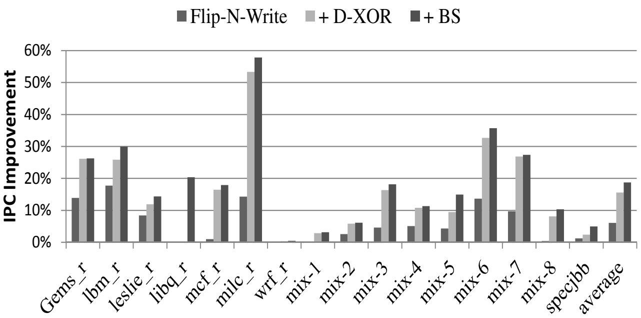

Figure 19: IPC improvement relative to H6 for ECC Memory.

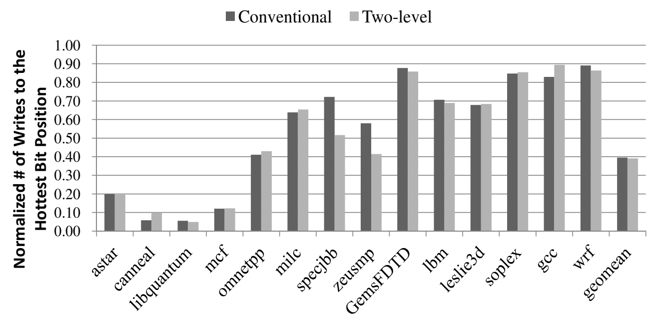

Figure 21: Comparison between conventional row shifting and two-level row shifting.

with ECC redundant bits. The results are normalized to ECC with H6. We first apply Flip-N-Write, which reduces the write service time to 86% of H6, on average. Then, we apply D-XOR to both data bits and ECC bits. For 256-byte write requests with 32-byte ECC, the mapping functions for data bits and ECC bits are L8^H8^H4 and L5^H5^H2. The write service time is further reduced to 54% of the H6 baseline. Lastly, we apply the bit swap function (BS) described in Section 4 to swap all 256 ECC bits with selected data bits. Compared to D-XOR, BS reduces write service time by 9% (average) to 49% of H6. When ECC bits are the bottleneck, BS reduces write service time by 61% for *libquantum*. BS is effective only if the average flip rate of ECC bits is much higher than the flip rate of the data bits. In *gcc* and *zeusmp*, BS degrades write performance as a result. It may be possible to adaptively disable BS in this situation.

6.3 Performance

Figure 19 shows IPC improvement over the H6 baseline. The graph shows the improvement for Flip-N-Write and our bit mapping functions for ECC PCM with 64 cell groups.

The results show that Flip-N-Write provides 6.1% performance improvement on average. Adding D-XOR, the improvement increases to 15.6%. *Libquantum* has a perfect cyclical pattern. Hence, D-XOR cannot improve over the baseline. BS provides an additional 3.1% improvement on average. Because *libquantum* has a few modified data bits, BS is particularly effective for *libq_r*, which has a 20% improvement in IPC. Combining D-XOR and BS achieves an extra 12% IPC improvement on average over Flip-N-Write. The IPC improvement is sensitive to PCM write traffic. Workloads with a high WPKI tend to have higher performance improvement. *Wrf* and *mix-1* have very small improvement because their write traffic is limited.

Figure 20 shows the average IPC improvement with different number of cell groups. We keep the maximum number of cell groups that can be concurrently accessed to be fixed. D-XOR is consistently better than the H6 baseline and Flip-N-Write. BS provides an additional small improvement, particularly when the number of cell groups is small. The figure also shows that careful selection of the number of cell groups is needed to achieve the best performance. If the number of cell groups is too large, fewer memory requests can be concurrently processed. If the number of cell groups is too small, the write service time becomes longer and the memory subsystem suffers a performance penalty from burst writes.

From the results in this section, we conclude that D-XOR mapping with BS and Flip-N-Write should be applied to achieve the best performance for write intensive workloads.

6.4 Intra-line Wear Leveling

For compatibility with bit mapping, we use two-level row shifting for intra-line wear leveling (see Section 4.6). To evaluate the effectiveness of our intra-line wear leveling scheme, we assume perfect inter-line wear leveling and simulate all write requests to a single memory line. The row shift offset is changed by one byte after every 256 writes [35]. To quantify the effectiveness of intra-line wear leveling, we measured the number of writes on the hottest bit position, which has the maximum number of writes. Figure 21 shows the number of writes normalized to the scheme without intra-line wear leveling. The figure shows that two-level row shifting is as effective as conventional row shifting on reducing the

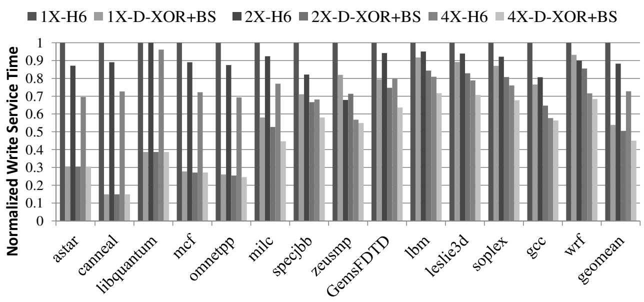

Figure 22: Average write service time for different division program widths normalized to H6 with 1X division program width (over all 14 benchmarks).

number of writes to the hottest bit position.

6.5 Impact of Division Program Width

Figure 22 shows the average write service time for our mapping function with different division program widths (averaged over all 14 benchmarks). All results assume that Flip-N-Write is enabled; the results are normalized to H6 with 1X division program width. The division program width is the maximum number of cells that can be programmed in one cell group. The mapping function used in the evaluation is D-XOR + BS. The figure shows that our mapping function consistently reduces write service time for different widths. However, the potential benefit decreases (on average) as the width is increased. As shown in Figure 18, *gcc* and *zeusmp* have a worse write service time with BS.

SET-to-RESET Latency	1.5:1	2:1	4:1	8:1
+Flip-N-Write	13.8%	13.3%	11.6%	10.3%
+Bit Mapping	50.7%	50.3%	49.2%	48.2%

Table 3: Reduction of write service time for different SET-to-RESET ratios.

6.6 Impact of SET to RESET Ratio

In the experiments above, the SET pulse is 150ns and the RESET pulse is 100ns [3]. We analyzed the sensitivity of our mapping functions to the SET:RESET latency ratio.

Table 3 shows the reduction of write service time with Flip-N-Write (one flip bit per 32 bits) and Bit Mapping (D-XOR + BS) for different SET latencies (RESET = 100ns). Our bit mapping technique is insensitive to the SET:RESET ratio. When SET:RESET ratio becomes larger, the improvement becomes slightly smaller because write service time is gradually dominated by SET programming time.

7. RELATED WORK

Recently, data encoding has been studied to improve PCM write performance and reduce PCM write energy. Cho and Lee proposed an adaptive bit flip technique that reduces the number of cells to program [2]. Zhang and Li proposed *data inversion and rotation* to store PCM data in a drift-tolerant format [33]. Xu et al. [28] proposed a data encoding scheme to reduce write energy. The data encoding problem is further studied in the context of MLC PCM [20, 11, 27]. Unlike these works that change the values of the data bits, our mapping functions change only the order of data bits. The data values are unchanged.

There is much work on studying memory values and memory access patterns. Yang and Gupta [30] studied *Frequent Value Locality*, which shows that a small number of distinct values tend to occur frequently in memory. Chen et al. [1] studied variations in memory spatial locality among different workloads. Somogyi et al. [24] observed strong spatial correlation of memory accesses at page granularity. Instead, our work studied the distribution patterns of modified data bits at memory line granularity.

XOR mapping has been utilized in other contexts. McFarling [17] uses XOR to combine branch address and global history for branch prediction. Gonzalez et al. [5] use XOR functions to reduce cache conflict misses. Zhang et al. [34] and Valero et al. [25] use XOR mapping for memory bank interleaving. Instead, our study shows that a D-XOR leads to a more balanced distribution of modified bits among PCM cell groups, which can substantially reduce PCM write service time and improve program performance.

Hay et al. [6] proposed a new scheme to schedule more outstanding write requests to multiple banks if each write request has a few modified bits. Our scheme reduces the write service time of individual write requests, which is orthogonal to their design. Yue and Zhu et al. [32] studied PCM writes with asymmetric RESET/SET programming widths. Our mapping function is still effective when asymmetric RESET/SET programming widths are enabled.

8. CONCLUSION

The mapping between data bits and PCM cell groups has a significant impact on PCM write service time, as we demonstrated in this paper. We first uncovered cyclical and cluster patterns in the distribution of modified bits in memory write requests. Based on observations about how bit writes are distributed, we showed that a balanced distribution of modified bits can be achieved by XOR mapping, catering to both cyclical and cluster patterns. We also proposed double XOR (D-XOR) mapping, which allows a single mapping function to be used for different PCM device programming widths. We also extended our technique to PCM with ECC.

Our results show that D-XOR reduces PCM write service time by 45% on average, which results in a 1.8× improvement in write throughput. Our best mapping function achieves an average 12% IPC improvement over Flip-N-Write for ECC-protected PCM. We conclude that data bit mapping is a simple and effective mechanism to increase PCM write throughput and program IPC.

9. ACKNOWLEDGMENTS

We acknowledge support from the PCM@Pitt research group. This research is supported partially by National Science Foundation grants CNS-1012070 and CCF-0811352.

10. REFERENCES

[1] C. F. Chen *et al.*, “Accurate and complexity-effective spatial pattern prediction,” in *Proc. of the 10th Int'l Symp. on High Performance Computer Architecture (HPCA)*, Feb. 2004.

[2] S. Cho and H. Lee, “Flip-N-Write: a simple deterministic technique to improve PRAM write

performance, energy and endurance," in *Proc. of the 42nd Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2009.

[3] Y. Choi *et al.*, "A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth," in *Int'l Solid-State Circuits Conference (ISSCC)*, Feb. 2012.

[4] T. J. Dell., "A white paper on the benefits of chipkill-correct ECC for PC server main memory." 1997, IBM Microelectronics Division.

[5] A. González *et al.*, "Eliminating cache conflict misses through XOR-based placement functions," in *Proc. of the 11th Int'l Conf. on Supercomputing (ICS)*, 1997.

[6] A. Hay *et al.*, "Preventing PCM banks from seizing too much power," in *Proc. of the 44th Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2011.

[7] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM Journal of Research and Development*, vol. 14, no. 4, Jul. 1970.

[8] E. Ipek *et al.*, "Dynamically replicated memory: Building reliable systems from nanoscale resistive memories," in *Proc. of the 10th Int'l Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2010.

[9] L. Jiang *et al.*, "LLS: cooperative integration of wear-leveling and salvaging for PCM main memory," in *Proc. of the 41st Int'l Conf. on Dependable Systems Networks (DSN)*, Jun. 2011.

[10] L. Jiang *et al.*, "FPB: fine-grained power budgeting to improve write throughput of multi-level cell phase change memory," in *Proc. of the 45nd Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2012.

[11] L. Jiang *et al.*, "Improving write operations in MLC phase change memory," in *Proc. of the 18th Int'l Symp. on High Performance Computer Architecture (HPCA)*, Feb. 2012.

[12] D. E. Kim, C. K. Kwak, and K. J. Lee, "Nonvolatile memory device and related methods of operation," 2011, U. S. Patent 7,876,609 B2.

[13] B. C. Lee *et al.*, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. of the 36th Int'l Symp. on Computer Architecture (ISCA)*, 2009.

[14] K.-J. Lee *et al.*, "A 90nm 1.8V 512Mb diode-switch PRAM with 266MB/s read throughput," in *Int'l Solid-State Circuits Conference (ISSCC)*, Feb. 2007.

[15] P. Magnusson *et al.*, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, Feb. 2002.

[16] O. Pearce *et al.*, "Quantifying the effectiveness of load balance algorithms," in *Proc. of the 26th Int'l Conf. on Supercomputing (ICS)*, Jun. 2012.

[17] B. Predictors and S. McFarling, "Combining branch predictors," 1993.

[18] M. Qureshi, M. Franceschini, and L. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *Proc. of the 16th Int'l Symp. on High Performance Computer Architecture (HPCA)*, Jan. 2010.

[19] M. K. Qureshi, "Pay-as-you-go: low-overhead hard-error correction for phase change memories," in *Proc. of the 44th Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2011.

[20] M. K. Qureshi *et al.*, "Morphable memory system: a robust architecture for exploiting multi-level phase change memories," in *Proc. of the 37th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2010.

[21] M. K. Qureshi *et al.*, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proc. of the 42nd Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2009.

[22] S. Schechter *et al.*, "Use ECP, not ECC, for hard failures in resistive memories," in *Proc. of the 37th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2010.

[23] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," in *Proc. of the 37th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2010.

[24] S. Somogyi *et al.*, "Spatial memory streaming," in *Proc. of the 33rd Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2006.

[25] M. Valero, T. Lang, and E. Ayguadé, "Conflict-free access of vectors with power-of-two strides," in *Proc. of the 6th Int'l Conf. on Supercomputing (ICS)*, 1992.

[26] C. Villa *et al.*, "A 45nm 1Gb 1.8V phase-change memory," in *Int'l Solid-State Circuits Conference (ISSCC)*, Feb. 2010.

[27] J. Wang *et al.*, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *Proc. of the 29th Int'l Conf. on Computer Design (ICCD)*, Oct. 2011.

[28] W. Xu, J. Liu, and T. Zhang, "Data manipulation techniques to reduce phase change memory write energy," in *Proc. of the 14th Int'l Symp. on Low Power Electronics and Design (ISLPED)*, 2009.

[29] B. D. Yang *et al.*, "A low power phase-change random access memory using a data-comparison write scheme," in *Proc. of the IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, May 2007.

[30] J. Yang and R. Gupta, "Frequent value locality and its applications," *ACM Trans. Embed. Comput. Syst.*, vol. 1, no. 1, Nov. 2002.

[31] D. H. Yoon *et al.*, "FREE-p: protecting non-volatile memory against both hard and soft errors," in *Proc. of the 17th Int'l Symp. on High Performance Computer Architecture (HPCA)*, Feb. 2011.

[32] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *Proc. of the 19th Int'l Symp. on High Performance Computer Architecture (HPCA)*, Feb. 2013.

[33] W. Zhang and T. Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," in *Proc. of the 42nd Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2009.

[34] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Proc. of the 33rd Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2000.

[35] P. Zhou *et al.*, "A durable and energy efficient main memory using phase change memory technology," in *Proc. of the 36th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2009.